ADP an Agent Decomposition Planner CoDMAP 2015

ADP an Agent Decomposition Planner CoDMAP 2015

Matthew Crosby
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
m.crosby@ed.ac.uk

Abstract

ADP (an Agent Decomposition-based Planner) is designed to deduce agent decompositions from standard PDDL-encoded planning problems and then to exploit such found decompositions to generate a heuristic used for efficient planning. The decomposition process partitions the problem into an environment and a number of agents which act on and influence the environment, but can not (directly) effect each other. The heuristic calculation is an adaptation of the FF relaxation heuristic to incorporate multiagent information. Relaxed planning graphs are only ever generated for single-agent subproblems. However, when cooperation is necessary, an agent's starting state may include facts added by others.

Introduction

ADP is a complete, satisficing (non-optimal) centralised planning algorithm that attempts to compute and utilise agent decompositions for the sole purpose of improving planning time. As such, it does not take into account common multiagent concerns such as privacy, trust or strategic considerations. It has been shown (Crosby, Rovatsos, and Petrick 2013; Crosby 2014) that useful decompositions can be found and successfully utilised in around forty percent of IPC domains (IPC 2011), a collection of domains which are not explicitly designed to be multiagent, yet contain some obviously multiagent settings.

The first section of this paper provides a brief high-level overview of the ADP algorithm, while the following section provides more detail including explicit discussion of the decomposition process and heuristic calculation. The third section presents a summary of the agent decomposition results for the domains used in CoDMAP 2015 after which some limitations and future plans for ADP are discussed. Technical details of the planner and other information relevant to CoDMAP 2015 can be found in the final section.

ADP Overview

ADP is split into two components, a decomposition phase and a heuristic calculation. The decomposition phase processes the planning problem and attempts to find a useful multiagent decomposition. As depicted in Figure 1, a decomposed domain is made up of a number of agents, each with an internal state, and an environment that the agents are acting upon. Actions that can influence or depend upon the internal state of an agent are assigned to only that agent. Actions that only involve the environment are assigned to all agents. The decomposition algorithm is guaranteed to return a decomposition in which no actions that influence (or rely upon) multiple agents' internal states exist. This means that agents can only influence themselves or the environment (thought the actions available to an agent in a given state may depend on how others have influenced the environment).

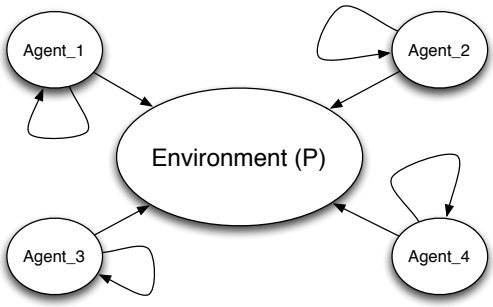

Figure 1: A depiction of an agent decomposition of the variables in a domain. Agents can only change the state of themselves and/or the environment.

When a decomposition is found, ADP calculates a heuristic value (used to guide a greedy best first search) that attempts to exploit the multiagent structure to find plans faster than the alternative single-agent approach. If no suitable decomposition can be found, then ADP defaults to the the single-agent FF heuristic (Hoffmann and Nebel 2001).

The main idea behind the multiagent heuristic calculation is to only ever generate planning graphs for a single agent subproblem at a time. In 'coordination points' each agent computes its heuristic value for each goal proposition by generating its relaxed planning graph from the current state. Cooperation is achieved (where necessary) by combining all individually reachable states and using this as input for successive rounds of individual relaxed planning graph generation. As a result of this process, a coordination point is as-

Algorithm 1: High-level Overview of ADP

Input : MPT $\langle V, I, G, A\rangle$
Output: Plan or $\perp$

Calculate Agent Decomposition Φ
$S \leftarrow I$
CoordPoint(S) [initialises $S.agent$, $S.goals$ and $S.macro$]
Greedy BFS from S using h_adp heuristic [When the successor of S is generated it copies $S.agent$, $S.goals$ and $S.macro$ from its predecessor]

Algorithm 2: Decomposition Algorithm

Input : MPT $\langle V, I, G, A\rangle$, Causal Graph CG, $\Phi = \{\}$
Output: Agent Decomposition $\Phi = \{\phi_1, \ldots, \phi_n\}$

$\Phi \leftarrow \{\{v\} : v \in V \wedge$ v root node of CG$\setminus$2-way cycles$\}$
repeat
  foreach $\phi_i \in \Phi$ **do**
    $\phi_i \leftarrow \phi_i \cup \{v \in V : v$ only successor of $\cup \phi_i\}$
  $\Phi \leftarrow \Phi$ where agents sharing joint actions are combined
until Φ *can no longer be refined*

signed an agent (the one with the most individually achievable goals), a set of agent goals (all propositions the assigned agent can achieve alone along with all propositions necessary for other agents to achieve goals requiring cooperation) and a macro heuristic value that estimates the coarse distance to the goal. At non-coordination points, the heuristic value is only updated by the currently assigned agent's progress towards its currently assigned goals.

ADP Details

Algorithm 1 gives a high-level pseudo-code overview of ADP. The algorithm takes a multi-valued planning task (MPT) calculated by the Fast-Downward Planning System (Helmert 2006) as input and consists of an initial preprocessing decomposition phase, followed by greedy best-first search using the ADP heursitic.

An MPT is represented by $\Pi = \langle V, I, G, A\rangle$, where:

- V is a finite set of state variables v, each with an associated finite domain D_v,
- I is a state over V called the initial state,
- G is a partial variable assignment over V called the goal, and
- A is a finite set of (MPT) actions over V.

In what follow we assume the standard definition of preconditions *pre(a)* and effects *eff(a)*.

Decomposition

The decomposition algorithm creates a partitioning of the variable set V into variable subsets ϕ_i for each agent i, which leaves a public variable set P that contains the variables that pertain to the environment. An overview of the decomposition algorithm is shown in Algorithm 2.

Algorithm 3: h_adp Calculation

Input : State S with $S.agent$, $S.goals$ and $S.macro$
Output: h_adp

$S.micro \leftarrow$ *hff*$(V|_{S.agent}, S|_{S.agent}, G|_{S.goals}, A|_{S.agent})$
if *S.micro == 0 or deadend* **then**
  CoordPoint(S)
  $S.micro \leftarrow$ *hff*$(V|_{S.agent}, S|_{S.agent}, G|_{S.goals}, A|_{S.agent})$
h_adp = S.macro + S.micro

First, a list of potential agents is found from the causal graph by taking every root node that exists in the graph once all two-way cycles have been removed. These root nodes must still have at least one successor node to be considered.

In the next step, the potential decomposition is refined by first extending agent sets to be as large as possible and then reducing the number of sets based on the existence of any joint actions. Agent sets are extended by recursively adding any node of the causal graph that is a successor of only variables already included in that agent set.

Actions in the domain can be categorised based on potential decompositions (partitions). An action is said to be *internal* to agent i for a given decomposition $\Phi = \{\phi_1, \ldots, \phi_n\}$ iff: $\exists v \in pre(a) : v \in \phi_i$, and$v \in pre(a) \rightarrow v \in \phi_i \cup P$. In other words, the preconditions of a must depend on an internal state variable of i and can only change i's internal state variables or the domain's public variables.

A *public action* is any action where: $v \in pre(a) \rightarrow v \in P$, i.e., where the preconditions do not depend on the internal state of any of the agents. An agent's action set is the set of all internal actions of that agent, denoted by Act_i.

An action is *joint* between agents i and j given decomposition $\Phi = \{phi_1, \ldots, \phi_n\}$ iff. $\exists v \in pre(a) : v \in \phi_i$, and$\exists v \in pre(a) : v \in \phi_j$. An action can be joint between multiple agents. In the second stage of the algorithm all agents involved in any joint actions are merged.

In a decomposition returned by ADP, the sets ϕ_i are guaranteed to have the *agent property*. In particular, a variable set ϕ_i (as part of a full decomposition Φ) has the agent property when for all $a \in A$ and variables $v \in V$:

$$v \in \phi_i \wedge v \in \mathit{eff}(a) \rightarrow a \in Act_i.$$

In other words, any agent variable can only be modified by an action of that agent. The proof of this can be found in (Crosby 2014).

Heuristic Calculation

The h_adp heuristic calculation is formed of two parts as shown in Algorithm 3. There is a global *coordination point* calculation that is performed infrequently and is used to pick out a single agent and a set of goals for that agent to attempt to achieve. There is also a micro single-agent heuristic calculation that is identical to that used by FF (Hoffmann and Nebel 2001) on the planning problem restricted to the current chosen agent and its current set of goals and subgoals.

Algorithm 4: Coordination Point Calculation

Input : State S
Output: $S.agent, S.goals$ and $S.macro$

```
Iterated Relaxed Planning Graph Generation
if Max layer > 0 then
    Calculate Subgoals
Assign Goals
S.agent ← agent with most goals with min h_add
S.goals ← all goals achievable by S.agent
S.macro ← N × |G \ S.goals|
```

Coordination Point Calculation An overview of the coordination point calculation is shown in Algorithm 4. It associates a chosen agent, a goal set and a macro heuristic value with the current state. This extra state information is carried over whenever a successor state is generated and (re)calculated whenever the current agent's goal set is completed or becomes impossible.

Iterated Relaxed Planning Graph Generation. Given a state, each agent generates their full relaxed planning graph for their restricted problem from that state. That is, each agent uses an iterative process to create a graph containing all possible actions it can perform (ignoring delete effects) and all possible propositions that can be reached by performing those actions.

It may happen that some propositions can only be reached if agents cooperate. For example, one agent may need to unlock a door before another can pass through. Because the agents create their own planning graphs (using only their own subproblems) they will not include any parts of the search space only reachable by cooperation. If not all goals are reachable by at least one agent after the first round of relaxed planning graph generation, the collected final state of all the agents is formed and used as input for a subsequent *layer* of relaxed planning graphs. Each successive iteration introduces a new *layer* with the first being *layer* 0. Repeating this process until no more states are added by any agent is guaranteed to cover every reachable state in the full (not decomposed) problem.

Calculate Subgoals: Any time a goal proposition appears for the first time in a layer above 0 this means that it cannot be reached by an agent on its own. In this case, subgoals are calculated. Plan extraction is used to find out which propositions are necessary from a previous layer in order to achieve each goal. These propositions are called subgoals. All subgoals from layer 0 are added as to the agent that achieved them first. Using the door example, if the second agent needs to pass through the door to achieve a goal, the subgoal of unlocking the door will be assigned to the first agent.

Assign Goals: The next part of the coordination point calculation chooses which agent is going to be performing the local search. First, each goal is assigned to the agent that can achieve it with the lowest estimated cost from the relaxed planning graphs. That is, it is assigned to the agent that added it with the lowest h_add value (Hoffmann and Nebel 2001). An agents goal set is then formed of all goals and subgoals assigned to it. The agent with the largest goal set is chosen as $S.agent$ along with all of its goals (and any subgoals that it may have been assigned).

As a final part of the coordination point calculation the value $S.macro$ is calculated. This is used to provide a global heuristic estimate of the distance through the overall search space. This is calculated as $N \times |G \setminus S.goals|$ where N is some large number chosen such that it dominates the single-agent FF heuristic value of a state.

Decomposition of CODMAP Domains

This section discusses the decompositions that ADP finds for the competitions domains used in CODMAP 2015. The results are shown in Table 1. The second column of the table 'Usable' reports whether or not ADP managed to find a usable decomposition for the domain. ADP found a suitable decomposition for nine of the twelve domains in the competition, failing to find one in blocksworld, driverlog and sokoban. This does not mean that no sensible decomposition exists for this domain, but that ADP could not find one that respects the *agent property* and contains no joint actions.

In general, decompositions returned by ADP are consistent across all problem instances for each particular domain. The one exception is Woodworking for which there was at least one problem instances in which ADP could not find a decomposition (represented by an asterisk in the table).

The third column of the table 'Joint' shows whether or not ADP found a decomposition that includes joint actions. The only domain for which this differs is Driverlog for which a decomposition was found (drivers+trucks) but was not used. There is no theoretical reason why the APD heuristic cannot be applied when joint actions exist, however in such domains, the algorithm tends to perform very poorly and much worse than if no decomposition is returned. Note that for some domains a decomposition including joint actions was found part-way through the decomposition algorithm, but the final decomposition did not include any after agents were merged.

The final three columns of the table compare the predefined decompositions for the problems to the decomposition that ADP finds. ADP found the same decomposition in exactly half of the ten remaining domains. In Driverlog and Taxis, ADP found a very similar decomposition including the trucks as well as the drivers in Driverlog and only including the taxis and not the passengers in Taxis. In Depot, ADP finds a completely different decomposition which treats the trucks as agents and also contains a separate agent that includes every single crate in the domain. The two remaining domains Woodworking and Wireless return fairly odd looking decompositions and it is expected that ADP will perform poorly on these domains.

ADP also found some extended agent sets not reported in the table. For example, in satellites the decomposition found by ADP includes the variables representing the state of the instruments' of each satellite with each individual satellite. In the Zenotravel domain the agent sets include the fuel levels for each plane.

Domain	Usable	Joint	Predefined Decomp	ADP Decomp	Match
blocksworld	✗	✗	agents	na	–
depot	✓	✓	drivers + dists + depots	trucks + crates*	✗
driverlog	✗	✓	drivers	drivers + trucks	✗
elevators	✓	✓	lifts(fast/slow)	lifts(fast/slow)	✓
logistics	✓	✓	trucks + airplanes	trucks + airplanes	✓
rovers	✓	✓	rovers	rovers	✓
satellites	✓	✓	satellites	satellites	✓
sokoban	✗	✗	players	na	–
taxi	✓	✓	taxis + passengers	taxis	✗
wireless	✓	✓	nodes + bases	messages by node + messages by base	✗
woodworking	✓(*)	✓	different tools	boards(available) + saw	✗
zenotravel	✓	✓	planes	planes	✓

Table 1: The decompositions found by ADP on the CODMAP problem domains.

Limitations and Future Work

This section briefly states some of the current limitations of ADP and current plans for improvement and extension of the planner in the future. The decomposition algorithm is known to return a decomposition with the agent property but it is currently unknown if it will always find such a decomposition if one exists. Further theoretical work and experimentation with different possible decomposition definitions is planned along with the release of a a standalone decomposer that can be used to find decompositions of PDDL-encoded planning problems.

The ADP heuristic calculation is based on the FF heuristic. However, there are a large number of other successful planning heuristics that have since been developed and it is likely that the overall multiagent approach can be applied to some of these techniques. It will also be interesting to extend ADP to include reasoning about action costs, numeric fluents and other extensions of PDDL or to include more multiagent aspects of the planning problems.

Finally, ADP is currently implemented as a single-threaded process. As each agent is always acting only working on their own internal problem, there is clearly scope for a multi-threaded version in which agents explore the search space independently.

ADP Details Summary

This final section presents a summary of the details of ADP relevant to CoDMAP 2015. ADP is a complete, satisficing (non-optimal) centralised single-threaded planning algorithm. ADP was implemented as a heuristic plug-in for the Fast-Downward planning system (Helmert 2006). The preprocessing of the planning problem into an MPT and the search algorithm are left unchanged. ADP simply calculates a decomposition and provides a heuristic value for each state that is queried during search and also stores a macro-heuristic value, agent, and goalset for each state. ADP is called with the option cost_type=2 and using the lazy_greedy search algorithm. Source code for ADP can be found online at the authors homepage.

ADP ignores the agent factorization presented in the MA-PDDL files, instead determining the agents present (if any) itself. The private/public separation is therefore also ignored. ADP does have an internal representation for private and public facts and actions used for decomposition calculation but does not use this directly for search.

Sometimes ADP will not be able to find an agent decomposition (defaulting to single-agent planning behaviour) or find a different decomposition as to that specified in the CoDMAP files. The details of these cases are explained in an earlier section of this paper.

Two versions of ADP were submitted to the CodMAP planning competition. Planner1 is the ADP implementation described in this paper. Planner2 is a legacy version of the code that is functionally identical except that instead of storing the macro-heuristic value and agent assignment, it (incorrectly) assumes that this can be carried over from the previously searched state. This legacy version was entered out of curiosity and the fact that it has produced some interesting results with some recent work in the planning community showing the possible value of including some element of randomness (essentially what the legacy version does) in the search process.

References

Crosby, M.; Rovatsos, M.; and Petrick, R. P. A. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 46–54.

Crosby, M. 2014. *Multiagent Classical Planning*. Ph.D. Dissertation, University of Edinburgh.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

IPC. 2011. http://www.plg.inf.uc3m.es/ipc2011-deterministic/. Web Site.